5. Implementing a Caesar Cipher Encryption and Decryption using Classes

Create a Ruby program using classes to encrypt and decrypt messages using the Caesar cipher technique.

- CaesarCipher Class: Define a class `CaesarCipher` with the following methods:
  - `initialize(shift)`: Constructor that initializes the shift value (shift) for encryption and decryption.
  - `encrypt(message)`: Method that takes a plaintext message (message) and returns the encrypted message using the Caesar cipher technique with the specified shift.
  - `decrypt(message)`: Method that takes a ciphertext message (message) and returns the decrypted message using the Caesar cipher technique with the specified shift.
  - `cipher(message, shift)` : Method in a Caesar cipher implementation encrypts or decrypts a given message string based on the shift value provided
    - Can only be called from within the same instance

```
┌──(imenⓀhbtn-lab)-[…/scripting_cyber/0x00-ruby_scripting]
└─$ cat 5-main.rb
require_relative '5-cipher'


cipher = CaesarCipher.new(5)

# Encrypting a message
plaintext = "Hello, Holberton!"
encrypted_message = cipher.encrypt(plaintext)
puts "Encrypted message: #{encrypted_message}"

# Decrypting the encrypted message
decrypted_message = cipher.decrypt(encrypted_message)
puts "Decrypted message: #{decrypted_message}"
```

```
┌──(imenⓀhbtn-lab)-[…/scripting_cyber/0x00-ruby_scripting]
└─$ ruby 5-main.rb
Encrypted message: Mjqqt, Mtqgjwyts!
Decrypted message: Hello, Holberton!
```

The **Caesar cipher** is one of the simplest and oldest encryption techniques. It is a **substitution cipher**

that shifts each letter in the plaintext by a fixed number of places in the alphabet.

Theory and Mechanism

1. **Substitution**:
   - Each letter in the plaintext is replaced by another letter located a fixed number of positions down (or up) the alphabet.
   - For example, with a shift of 3:
     - A → D
     - B → E
     - Z → C (wrapping around the alphabet).
2. **Mathematical Representation**:
   - Let each letter in the alphabet correspond to a number (`A = 0`, `B = 1`, ..., `Z = 25`).
   - For **encryption**:
     (E(x) = (x + n) \mod 26)
     where:
     - (x) is the numeric position of the letter in the alphabet.
     - (n) is the shift value.
     - (\mod 26) ensures the result wraps around within the range of the alphabet.
   - For **decryption**:
     (D(x) = (x - n) \mod 26)
3. **Key**:
   - The **shift value** (e.g., 3) is the key in the Caesar cipher. Both the sender and receiver must know the key to encrypt and decrypt the message.
4. **Example**:
   - Plaintext: `HELLO`
   - Shift: `3`
   - Ciphertext: `KHOOR`
5. **Non-Alphabet Characters**:
   - Non-alphabet characters (spaces, punctuation, numbers) are often left unchanged in the encryption and decryption process.

Strengths of the Caesar Cipher

- **Simplicity**: Easy to implement and understand.
- **Historical Significance**: Used by Julius Caesar to securely communicate with his generals.

Weaknesses of the Caesar Cipher

1. **Vulnerability to Brute Force**:
   - The Caesar cipher only has **25 possible keys** (shifts), making it easy to crack by trying all possible shifts.
2. **Pattern Preservation**:
   - The structure of the plaintext is preserved in the ciphertext, revealing potential clues to an attacker.
3. **Letter Frequency Analysis**:
   - In languages like English, certain letters appear more frequently (e.g., `E` is common). Attackers can use this knowledge to guess the shift.

Modern Relevance

The Caesar cipher is no longer used for secure communications but is an excellent educational tool for understanding the basics of cryptography. It forms the foundation for more advanced techniques, such as **substitution ciphers** and **modern cryptographic algorithms**.

Sure! Here's a step-by-step explanation of the **Caesar cipher code** for the `CaesarCipher` class:

Code Structure

The class has the following components:

1. **Constructor (`initialize`)**: Sets the shift value.
2. **Public methods (`encrypt`, `decrypt`)**: For encryption and decryption.
3. **Private method (`cipher`)**: Performs the core logic for shifting characters.

Here's the code:

```
class CaesarCipher
  def initialize(shift)
    @shift = shift  # Store the shift value
  end

  def encrypt(message)
    cipher(message, @shift)  # Call the cipher method to shift characters forward
  end

  def decrypt(message)
    cipher(message, -@shift)  # Call the cipher method to shift characters backward
```

```
  end

  private

  def cipher(message, shift)
    # Map each character in the message
    message.chars.map do |char|
      if char.match?(/[A-Za-z]/)  # Check if the character is a letter
        base = char.ord < 91 ? 'A'.ord : 'a'.ord  # Determine if it's
uppercase or lowercase
        ((char.ord - base + shift) % 26 + base).chr  # Shift and wrap around
      else
        char  # Leave non-alphabetic characters unchanged
      end
    end.join  # Combine the array back into a string
  end
end
```

Step-by-Step Explanation

1. Constructor (`initialize`)

```
def initialize(shift)
  @shift = shift  # Store the shift value as an instance variable
end
```

- When you create an instance of `CaesarCipher`, you pass a **shift value** (e.g., `5`).
- The shift value determines how many places to move each letter in the alphabet.

---

2. Encrypt Method

```
def encrypt(message)
  cipher(message, @shift)  # Call the cipher method with the shift value
end
```

- Takes a plaintext message (e.g., `"Hello, World!"`).
- Calls the private `cipher` method to process the message, applying the positive shift value.

---

3. Decrypt Method

```
def decrypt(message)
  cipher(message, -@shift)  # Call the cipher method with the negative shift
value
end
```

- Takes an encrypted message (e.g., `"Mjqqt, Btwqi!"`).
- Calls the `cipher` method with the **negative shift**, effectively reversing the encryption.

---

4. Cipher Method

The core logic for both encryption and decryption is handled here:

```
def cipher(message, shift)
  message.chars.map do |char|  # Split the message into characters and process each one
    if char.match?(/[A-Za-z]/)  # Check if the character is a letter
      base = char.ord < 91 ? 'A'.ord : 'a'.ord  # Determine ASCII base ('A' or 'a')
      ((char.ord - base + shift) % 26 + base).chr  # Perform the shift
    else
      char  # Leave non-alphabetic characters unchanged
    end
  end.join  # Combine the transformed characters back into a single string
end
```

Breaking it Down:

1. **Iterating through Characters**:
   - `message.chars.map`: Splits the string into an array of characters, then processes each one individually.
2. **Checking Alphabet Characters**:
   - `char.match?(/[A-Za-z]/)`: Determines if the character is a letter (uppercase or lowercase).
   - If it is not a letter (e.g., punctuation, spaces), the character is returned unchanged.
3. **Shifting Logic**:
   - **Determine ASCII Base**:

     ```
     base = char.ord < 91 ? 'A'.ord : 'a'.ord
     ```

     - If the character is uppercase (`A-Z`), the base is ASCII value `65` (`'A'.ord`).
     - If the character is lowercase (`a-z`), the base is ASCII value `97` (`'a'.ord`).
   - **Shift and Wrap**:

     ```
     ((char.ord - base + shift) % 26 + base).chr
     ```

     - `char.ord`: Converts the character to its ASCII value.
     - `char.ord - base`: Normalizes the character to a 0-25 range (`A=0`, `B=1`, ..., `Z=25`).
     - `+ shift`: Applies the shift value.

    - `% 26`: Ensures the result wraps around if it goes past `Z` (for encryption) or before `A` (for decryption).
    - `+ base`: Converts back to the original ASCII range (`65-90` for uppercase, `97-122` for lowercase).
    - `.chr`: Converts the ASCII value back to a character.
4. **Joining the Transformed Characters**:
    - `join`: Combines the processed characters into a single string.

Example Walkthrough

Encrypting "Hello, World!" with a shift of 5:

- `H` → `M`
  ((72 - 65 + 5) % 26 + 65 = 77)
- `e` → `j`
  ((101 - 97 + 5) % 26 + 97 = 106)
- `l` → `q`
- `o` → `t`
- Non-alphabetic characters (`,`, ` `, `!`) remain unchanged.

Result: `"Mjqqt, Btwqi!"`

Decrypting "Mjqqt, Btwqi!" with a shift of 5:

- `M` → `H`
  ((77 - 65 - 5) % 26 + 65 = 72)
- `j` → `e`
  ((106 - 97 - 5) % 26 + 97 = 101)

Result: `"Hello, World!"`

Summary

- **`initialize`**: Sets the shift value.
- **`encrypt`**: Calls `cipher` with a positive shift.
- **`decrypt`**: Calls `cipher` with a negative shift.
- **`cipher`**: Implements the Caesar cipher logic for both encryption and decryption.
- Handles uppercase, lowercase, and non-alphabetic characters seamlessly.